CEAS Space Journal
https://doi.org/10.1007/s12567-021-00371-7

ORIGINAL PAPER

ScOSA system software: the reliable and scalable middleware for a heterogeneous and distributed on-board computer architecture

Andreas Lund[1] · Zain Alabedin Haj Hammadeh [2] · Patrick Kenny[1] · Vishav Vishav[2] · Andrii Kovalov[2] · Hannes Watolla[2] · Andreas Gerndt[2,3] · Daniel Lüdtke[2]

Received: 1 February 2021 / Revised: 1 April 2021 / Accepted: 11 May 2021

Abstract
Designing on-board computers (OBC) for future space missions is determined by the trade-off between reliability and performance. Space applications with higher computational demands are not supported by currently available, state-of-the-art, space-qualified computing hardware, since their requirements exceed the capabilities of these components. Such space applications include Earth observation with high-resolution cameras, on-orbit real-time servicing, as well as autonomous spacecraft and rover missions on distant celestial bodies. An alternative to state-of-the-art space-qualified computing hardware is the use of commercial-off-the-shelf (COTS) components for the OBC. Not only are these components cheap and widely available, but they also achieve high performance. Unfortunately, they are also significantly more vulnerable to errors induced by radiation than space-qualified components. The *ScOSA (Scalable On-board Computing for Space Avionics) Flight Experiment* project aims to develop an OBC architecture which avoids this trade-off by combining space-qualified radiation-hardened components (the reliable computing nodes, RCNs) together with COTS components (the high performance nodes, HPNs) into a single distributed system. To abstract this heterogeneous architecture for the application developers, we are developing a middleware for the aforementioned OBC architecture. Besides providing an monolithic abstraction of the distributed system, the middleware shall also enhance the architecture by providing additional reliability and fault tolerance. In this paper, we present the individual components comprising the middleware, alongside the features the middleware offers. Since the *ScOSA Flight Experiment* project is a successor of the *OBC-NG* and the *ScOSA* projects, its middleware is also a further development of the existing middleware. Therefore, we will present and discuss our contributions and plans for enhancement of the middleware in the course of the current project. Finally, we will present first results for the scalability of the middleware, which we obtained by conducting software-in-the-loop experiments of different sized scenarios.

Keywords Fault-tolerance · On-board computer architecture · Embedded systems · Middleware · Distributed systems · Reliable computing

✉ Andreas Lund
Andreas.Lund@dlr.de

Zain Alabedin Haj Hammadeh
Zain.HajHammadeh@dlr.de

Patrick Kenny
Patrick.Kenny@dlr.de

Vishav Vishav
Vishav.Vishav@dlr.de

Andrii Kovalov
Andrii.Kovalov@dlr.de

Hannes Watolla
Hannes.Watolla@dlr.de

Andreas Gerndt
Andreas.Gerndt@dlr.de

Daniel Lüdtke
Daniel.Luedtke@dlr.de

[1] Institute for Software Technology, German Aerospace Center (DLR), Münchener Str.20, 82234 Weßling, Germany

[2] Institute for Software Technology, German Aerospace Center (DLR), Lilienthalplatz 7, 38108 Braunschweig, Germany

[3] Center for Industrial Mathematics (ZeTeM), University of Bremen, Bibliothekstr. 5, 28359 Bremen, Germany

Published online: 31 May 2021

1 Introduction

Software for unmanned spacecraft differs in one crucial point to most common embedded software: as soon as the mission launches, the direct maintenance of the embedded device is no longer possible. Therefore, spacecraft software has to be more reliable than common software, not only because of the maintenance problem, but also due to the high costs of a spacecraft mission and the extended risk of potential catastrophic consequences following a failure. Furthermore, the operational environment of a spacecraft offers some additional challenges to integrated circuits. Due to the lack of a protective magnetic field, the circuits are directly exposed to cosmic rays. This radiation, entering the device, can lead to a series of events in the electronics, called *soft errors*[1]. These in turn can trigger a multitude of different problems in the operative software of the spacecraft up to the complete loss of the system. The problems with the difficult environment on the one hand and the potential consequences of a failure of the system on the other hand motivate the implementation of reliable systems for spacecraft.

To achieve a reliable system, the system engineers often use space-qualified hardware. This hardware (e.g., the RAD750 from BAE Systems [3]) is radiation hardened and often comes with built-in low level fault-tolerance mechanisms. Unfortunately, the space qualification comes with a price: the performance is subpar in comparison to modern desktop architecture and standard embedded platforms (e.g., Xilinx Zynq-7000 [28]). This lack of performance in space-qualified hardware is a problem when it comes to processing-intensive applications (e.g., high-resolution observation), multi-application scenarios or autonomous spacecraft (e.g., rover missions on distant celestial bodies). Therefore, system engineers are increasingly using commercial-off-the-shelf (COTS) components to build their spacecraft. This increases the performance while reducing the costs but is accompanied by a decrease of reliability. Overcoming this trade-off between reliability and performance is a current research interest in the field of space systems engineering.

The *German Aerospace Center (DLR)* started researching a solution for this topic in 2012 which became the *On-Board Computer—Next Generation (OBC-NG)* project [19] in 2013, continuing in 2017 with the *Scalable On-Board Computing for Space Avionics (ScOSA)* project [26]. The result of those two projects was an on-board computer architecture offering reliability combined with performance by mixing radiation-hardened hardware with COTS components and abstracting it via software to a monolithic system. The system architecture consists of independent, interconnected nodes (either reliable components or high-performance COTS components), each instantiating their own operating system, abstracted to a monolithic execution platform by a middleware. The system supports multiple applications running concurrently. The applications will be divided into channels (containing the states of the application) and tasks by the developer using the provided interface to the execution platform (called *Distributed Tasking*). The tasks and channels will then be mapped to the nodes of the system by the system designer. The middleware offers features for adjusting the task mapping for different mission phases as well as in case of a node failure. This is called reconfiguration of the system. Additionally, the middleware provides common services well known in the research area of *fault detection, isolation and recovery (FDIR)*. These services include: a voter service which implements a *Triple Modular Redundancy* [18] and the checkpointing service for a distributed state restoration after a task or node failure. The middleware is capable of operating on a heterogeneous set of processing nodes (either reliable or high-performance COTS components). This also includes a heterogeneous network architecture consisting of either Ethernet, SpaceWire [24] or a combination of both. Furthermore, the middleware is capable of supporting different operating systems (currently RTEMS and Linux) on the different nodes.

Following the ScOSA project, a successor project was launched. This project, called ScOSA Flight Experiment, began in January 2020. As the name suggests, this project aims to achieve a higher *Technical Readiness Level (TRL)* by preparing the in-orbit demonstration of the developed ScOSA on-board computer.

Alongside further improvements of the entire on-board computer, the middleware shall be enhanced for the demonstration. This includes enhancements to increase the robustness of the network stack, further *FDIR* techniques and services, and a restructuring of the developer API along with the integration of a new version of the tasking execution platform.

Additionally, the project shall contribute to the scientific area of COTS components in space. The focus in that context lies on the fault-tolerance mechanisms of the middleware as well as on the usage of COTS components and their induced risks to the mission in general.

In this paper, we present the ScOSA Flight Experiment, the ScOSA middleware, and our technical and scientific objectives for the project and first results for the overhead of the reconfiguration mechanism of ScOSA when scaled.

The remainder of this paper is structured as follows: In Sect. 2, we present the preliminary work together with related work for the distributed middleware. Section 3 gives a brief overview of the ScOSA Flight Experiment project, including its duration, milestones, and goals. Afterwards, in Sect. 3.2, we explain each different software component

[1] In fact, the radiation can cause several other types of errors as well, but for this paper, we will focus on soft errors.

building the middleware in detail. This is followed by an overview of the technical and scientific objectives for the ScOSA middleware (see Sect. 4). The paper ends with the presentation of some first experimental results for the scalability in Sect. 5 and a conclusion including an outlook to future work in Sect. 6.

2 Related work

The *ScOSA Flight Experiment* presented in this work builds on its predecessor projects, *ScOSA* [23, 26] and *OBC-NG* [5, 19]. In addition to the preliminary work of *DLR*, some related work in the field of fault-tolerance for spacecraft OBCs exists.

The research for fault-tolerant and distributed OBCs for spacecraft goes back several decades [21]. Often, a reliable OBC is implemented by cost-intensive hardware redundancy concepts, like *triple modular redundancy* [18].

More recently, the system designers tend to implement fault-tolerance more and more using software, especially in the middleware. Many of those rely on the CORBA standard [20] for distributed systems, for example *FLARe* [4], which is a middleware developed in the context of the *Lw-FT-RT-CORBA* standardization effort. *FLARe* is aiming to manage (soft) real-time tasks and fault-tolerance together by a decentralized resource monitor.

Another middleware enabling fault-tolerance based on *CORBA* is *MicroQoSCORBA (MQC)* [8]. It enables typical fault-tolerance mechanisms, like checksums, redundancy, and logical time stamping on application level.

Besides the general interest in fault-tolerance middleware for distributed embedded systems, this topic became increasingly relevant for the space domain in recent years. Afonso et al. for example, developed a framework upon a real-time embedded system for spacecraft [1]. The goal of the framework is to provide the applications with fault-tolerance mechanisms like *Recovery Blocks (RB)*, *Distributed Recovery Blocks (DRB)*, *TMR* and *N-version programming (NVP)*.

In [10], Fayyaz et al. also present a middleware for spacecraft which is distributed and fault-tolerant. The authors implement their middleware by instantiating a special programmed logic block (called *Adaptive Middleware for Fault-Tolerance AMFT* block) on each computing unit. The AMFT block constantly monitors the computing unit and communicates with all other AMFT blocks in the system. When a computing unit failure is recognized, the AMFT block informs all other blocks of this failure. A certain master computing unit will then redistribute all tasks to the remaining processors in the system.

Similar to this work and similar to our work, *NASA* developed their own architecture, called *High Performance, Dependable Multiprocessors* [30]. The architecture consists of one to two reliable processors, acting as system controllers and up to *N* independent COTS Field Programmable Gate Arrays (FPGAs). Similar to our approach, the authors implemented a fault-tolerant middleware. The middleware differs in terms of the reconfiguration after a node failure from our approach. The *Job Manager* of the middleware distributes the tasks among the available nodes on-line according to a load-balancing strategy. Our approach features precompiled, static configurations for node failures.

In [9], Dubey et al. designed a custom full-stack solution platform for fractionated spacecraft. This includes the operating system, a communication middleware, and a distributed fault manager.

Hecht et al. developed an adaptive fault-tolerant middleware for deep space missions in [12]. The adaptivity is derived from the ability to switch the fault tolerance mechanisms depending on the current environmental conditions and the available resources.

The aforementioned work on fault-tolerant middleware for spacecraft comes with many important features, as the provision of fault-tolerance mechanisms for the application developer, a heterogeneous architecture (with reliable and programmable nodes), a reconfiguration mechanisms, a distributed fault manager, and an adaptive middleware. Nevertheless, to the best of our knowledge, there is no work combining these features and supporting a system consisting of reliable and high-performance COTS components, yet.

3 The ScOSA flight experiment

The ScOSA Flight Experiment project aims to increase the *TRL* of the ScOSA OBC by means of an in-orbit demonstration. The targeted mission for the in-orbit demonstration is planned to be the next compact satellite (see Fig. 1 for an image of the last compact satellite by DLR) mission. The ScOSA OBC will be integrated to the compact satellite as a non-mission-critical secondary payload. The launch of the mission is planned to be in 2024.

3.1 ScOSA architecture

ScOSA is a new OBC architecture providing the features of fault-tolerance, high-performance, scalability, heterogeneity and a distributed execution of applications. It consists of a combination of RCNs with HPNs (in our case Zynq-7000 cores [28]), connected by SpaceWire [24] or Ethernet (see Fig. 2). Furthermore, sensors and actuators are connected into the system by so-called *Interface nodes*, which interact as interfaces to the peripherals for the RCNs and HPNs. Usually, several SpaceWire routers will be implemented to coordinate the communication between all nodes, but direct-linked nodes could also be used. To recognize node failures,

Fig. 1 DLR's last CompactSat: Eu:CROPIS [7, 16] (*Image:DLR, CC-BY 3.0*)

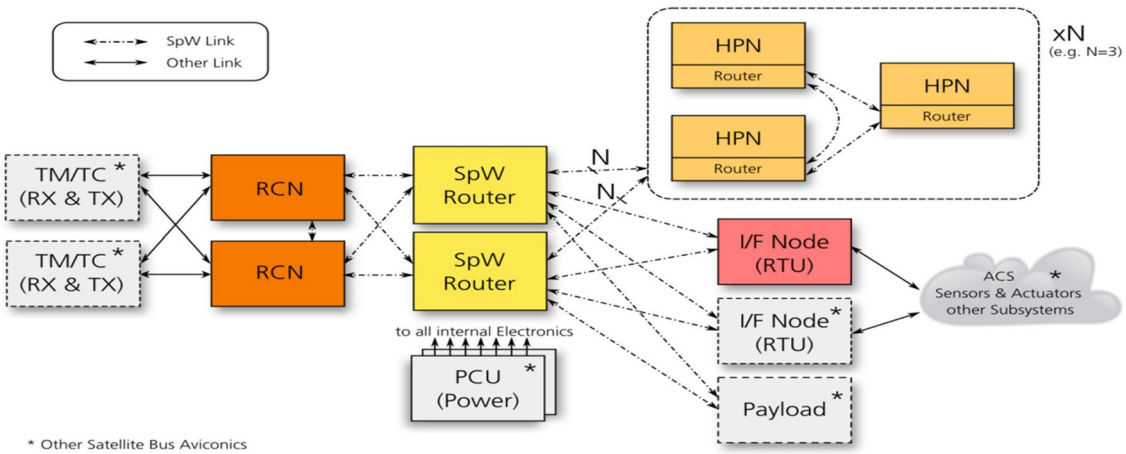

Fig. 2 Example of ScOSA's system architecture: Two reliable nodes (RCNs) connect the system to the telecommand and telemetry units. The RCNs are connected by SpaceWire routers to *N* high-performance nodes and the interface nodes which connect the sensors and actuators

the system implements a hierarchy between nodes and gives them one of three roles: *coordinator, observer* and *worker*. At each point in time, the system will have exactly one *coordinator* node which is responsible for monitoring all other nodes (by requesting heartbeat messages). All other nodes will become *worker*. Additionally, one or two nodes will become *observer* nodes, monitoring the *coordinator*.

3.2 Middleware architecture

In the following chapter, we explain the middleware and its features by means of its components comprising the middleware. Additionally, we explain the underlying thread model of the middleware. The middleware consists of three main components:

- a distributed execution platform, called *Distributed Tasking*
- a set of management services enabling the main *FDIR* features of the middleware
- a network stack for reliable messaging via SpaceWire or Ethernet

Those components are ordered into the layered architecture of the software stack (see Fig. 3) and will be explained further in the following subsections, beginning from the bottom with the network protocol.

3.2.1 SpaceWire-IPC

SpaceWire-IPC can be described as part of the transport layer (Layer 4 in ISO/OSI model) of the ScOSA system. The

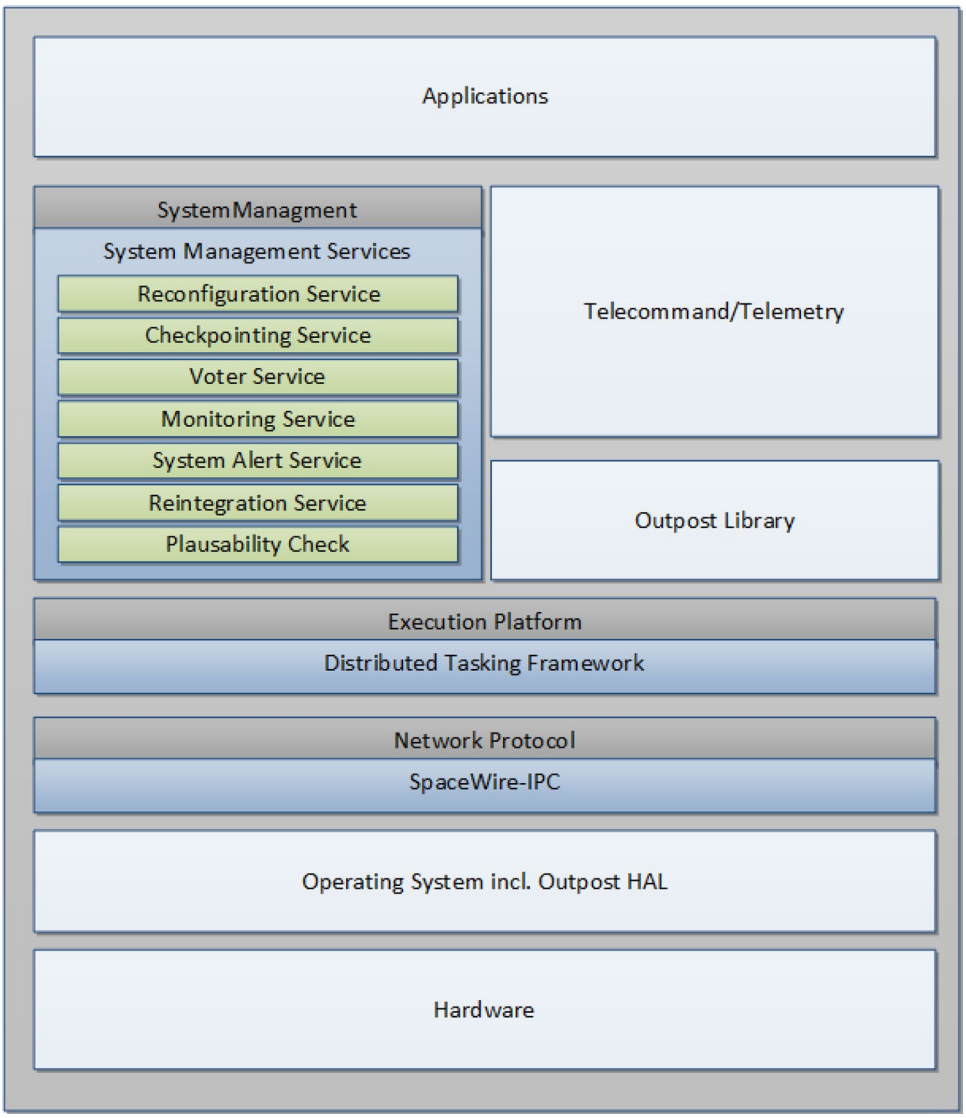

Fig. 3 ScOSA's layered software stack. The middleware consists of the three blue modules: System Management Services, Distributed Tasking Framework and the SpaceWire-IPC protocol. The system management in turn consists of several services providing fault-tolerance features to the system and the applications

main purpose of this layer is to extend the SpaceWire communications standard with means of inter-process communication (IPC) among nodes and management information to and from the *coordinator* node. The central paradigms of *SpaceWire-IPC* are reliable messages, error detection and error handling. It provides guaranteed delivery services and a timeout mechanism for reliable message transmissions. Even though its name suggest SpaceWire as transport medium, it is designed to support Ethernet, too.

3.2.2 Distributed tasking

The ScOSA middleware depends on an open-source multi-threading execution platform and a software development framework which is called *Tasking Framework* [11].

Using the *Tasking Framework*, applications are implemented as a graph of *tasks* that are connected via *channels*, and each task has one or more *inputs* to connect the tasks with the channels. However, the *Tasking Framework* does not have a direct support for distributed systems in which several tasks and channels may be mapped to one node, while other tasks and channels are mapped to another node. Therefore, the ScOSA middleware extends the *Tasking Framework* to the *Distributed Tasking Framework*.

3.2.3 System management services

The system management services implement the *FDIR* techniques of the middleware. It is important to understand that these services act as internal providers of functionality for the middleware itself (e.g., the reconfiguration service) or the application developer (e.g., the voter service). They are not meant to provide services to the outside, such kind of in-orbit services can be developed using the *ScOSA* middleware. The services will be instantiated as threads and connected to the network stack in order to communicate with the

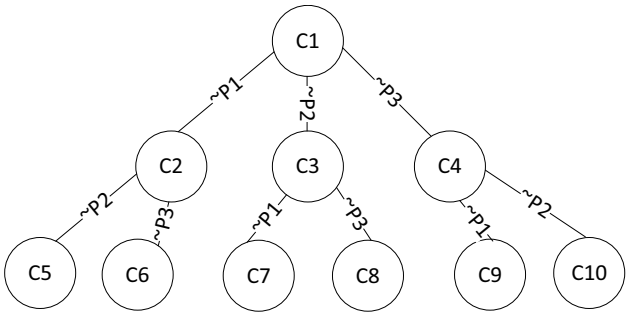

Fig. 4 This example shows several configurations (mapping of tasks to processing nodes) denoted as *C1–C10*. At the beginning, three processing nodes *P1–P3* are available. If one these processing nodes fails, a different configuration (without the failed node) has to be selected. The edges from a configuration show which configuration has to be chosen. $\backsim P_i$ depicts the failure of processing node P_i

other instances on other nodes. Additionally, this software component is responsible for storing the current configuration parameter and settings of the system, as well as the nodes' states and roles (*coordinator, observer, worker*). In the following we introduce the most important services in more detail.

Reconfiguration service This service is responsible for the reconfiguration of the task to node mapping (a.k.a. configuration) in case of a planned mission phase transition or a node failure. It maintains all configurations, which were created at design time by the system designer. In Fig. 4, one can see a reconfiguration tree in case of processing node failures. Each node of the tree represents a configuration. The edges of the tree point to configurations to be selected when a specific processing node P_i fails. Such configuration transitions are reported to the monitoring service which will shut down all tasks and channels on its node immediately. Afterwards, it starts the tasks and channels listed for its node in the new configuration.

Monitoring service The monitoring service is one of the key services for the reconfiguration feature of the middleware. It uses a heartbeat mechanism to monitor the state of the nodes in a distributed way. Its exact function depends on the role of the node. The different roles are: *coordinator*, *observer* and *worker*. *Coordinator* and *observer* nodes have additional responsibilities as part of the monitoring service, while *worker* nodes are only responsible for processing application tasks. Multiple nodes can have the roles *observer* and *worker*, but exactly one *coordinator* node exists at any time. Additionally, *observer* nodes need to implement a hierarchical order among themselves, such that there is an *observer1*, *observer2* and so on. The monitoring service of the *coordinator* node periodically sends messages to all other monitoring services on the other nodes. These messages are called *heartbeat requests*. A receiving monitoring service on a *worker* or an *observer* node will react with an acknowledgment message to this *heartbeat request*. When receiving the acknowledgment message, the *coordinator* node knows that the particular node is still alive and responding. If it does not receive an acknowledgment message from a node after a certain timeout and a resend of the *heartbeat request*, the *coordinator* node assumes that this node has failed. The monitoring service will then report that failure to the reconfiguration service, which will initiate a reconfiguration of the tasks to maintain the operational state of the system. To avoid that the *coordinator* node becomes the single-point-of-failure for the system, the first *observer* will request a heartbeat from the *coordinator* node periodically, too. The second *observer* node in turn will send *heartbeat requests* to the *coordinator* and the first *observer*. This scheme continues with every further *observer* in the system. How many *observers* a system implements is up to the system designer.

Voting service The voting service implements the concept of *triple modular redundancy (TMR)*, which is often used in the aerospace domain. With *TMR*, the same processing operation will either be executed three times sequentially on a single node or executed three times concurrently on separate nodes. Afterwards the result of the three operations will be compared and the majority result will be forwarded. The voting service implements the comparison and forwarding part, with the option to inform another service or application in case of any disagreement between the three results.

3.3 Thread model

The ScOSA middleware is implemented in a multi-thread approach, to utilize the capabilities of modern processor architectures. Additionally, extending the middleware with more services is easier when the functionality can be encapsulated into threads. The middleware in its current state invokes eight threads (see Fig. 5). While most of the threads are activated sporadically, the checkpointing thread and the monitoring thread can be configured with a specific period.

Note that number of *Tasking Framework* executor thread can be specified by the application developers and, therefore, varies from scenario to scenario.

4 Technical and scientific objectives

In the context of the in-orbit demonstration of the *ScOSA* OBC, we intend to enhance the fault-tolerant middleware and increase its stability by updating its components. The launch of the aimed compact satellite mission is planned for 2024. The *ScOSA Flight Experiment* project started in January 2020 and is planned to be finished end of 2022. During those three years, we plan to elaborate technical objectives for the *ScOSA* system software stack and contribute to

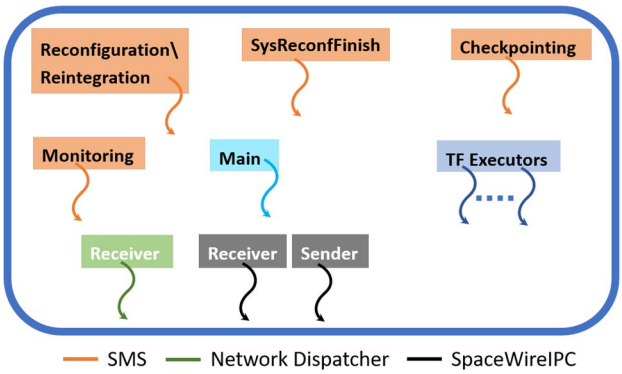

Fig. 5 The ScOSA middleware threads on one node. SMS: System Management Services, TF: *Tasking Framework*. All nodes have the same number of threads; however, the functionalities that will be executed by the threads depends on the node role, i.e., coordinator, observer, or worker. The tasking framework may have at least one executor thread

the scientific areas of fault-tolerance and COTS systems in space. In the following, we present an excerpt of the technical and scientific objectives:

4.1 New FDIR technique

As an enhancement, we plan to design, implement and evaluate a new service for the application developers, which inherits the transaction management of databases and the transactional memory in processors (e.g. Intel's Transactional Synchronization Extensions [13]). The idea is to process one operation and store its changes to the data in a special buffer, meanwhile a duplicate of the operation will be executed time-shifted to the first one, operating on the same original data and storing its data only temporarily. If the duplicate reaches a defined control value, it will compare its calculated value with the one from the first process. If the values are the same, the changes in the data will be applied to the real data set. If the values differ, a rollback will be initiated, which means that the data change buffer will be emptied and the operation will be rewound back to the beginning. If after several attempts the values still differ, an alert will be triggered. This is a similar approach to the *TMR* concept and the voting service with the difference that it leaves the actual data set untouched until the operation was confirmed.

4.2 Configuration modeling

One of the key features of *ScOSA* is the ability to apply new task configurations in case of a node failure. *ScOSA* uses a static approach for the reconfiguration, which means that all configurations are pre-compiled by the designer at design time. In case of a node failure during runtime, *ScOSA* searches the appropriate configuration and applies it. In case it cannot find a pre-compiled configuration for this particular combination of available nodes (*node state scenario*), it switches into the *safe mode*. The *safe mode* is a special operating mode in which the spacecraft will suspend all payloads and wait on ground interaction to solve the failure. To avoid too much ground interaction, it is, therefore, reasonable to provide a configuration for every possible node state scenario. Unfortunately the amount of node state scenarios grows exponentially with the amount of processors $|P|$ in the system and is equal to $2^{|P|}$. Taking an example of 8 processors, this already leads to 256 node state scenarios and necessary configurations. In that case, creating the configurations by hand is not an option any more. Therefore, we are providing a modeling tool which generates the configurations automatically. To provide good tasks-to-nodes mappings for the scenarios, the tool shall try to optimize two criteria: the utilization of every processor and the network traffic between the processors. Finding optimized configurations for the possible node state scenarios is a NP-complete problem; therefore, two different solvers are developed: an exact optimization solver for small-sized scenarios and a genetic algorithm for larger scenarios. These solvers will be then used to find optimal configurations.

4.3 Overhead of the reconfiguration system

Reconfigurable systems contribute to space systems in two ways. On the one hand, a reconfigurable system is more fault-tolerant compared to a statically configured system. On the other hand, a reconfigurable system can be used to support several mission phases reutilizing the hardware resources. This benefit comes at a cost: the cost of additional processing time for reacting to a failure or phase transition and the processing time for coordinating the reconfiguration. Due to the higher complexity of those algorithms, the validation and verification of such space software becomes more complex, too. To broaden the acceptance for reconfigurable systems in space, we strive to provide a reconfiguration module that generates acceptable overhead and scales with the system within a reasonable factor. For that reason, we will conduct a methodological runtime analysis on the failure recognition and the reconfiguration procedure of the *ScOSA* system.

4.4 COTS in space

One of the driving arguments for space-qualified processors in missions is their resistance against radiation-induced soft errors and failures, whereas COTS components are known to be vulnerable. For this reason, the use of COTS components is often secured by redundancy. To determine the needed redundancy for a system, it is important to know

the failure rate in space of the used COTS components. The ScOSA Flight Experiment will use the Zynq-7000 SoC [28] as high-performance nodes. This COTS component is often used for space missions nowadays [14, 15, 22], but only a few evaluations of its failure rates in space have so far been conducted [27]. Instead, there exist some studies of failure rates in ground-based particle accelerators [2, 6, 17, 25, 29]. We would like to contribute to this research field by providing failure rates for the Zynq-7000 SoC from the flight experiment in space. For this reason, we will design and implement an application which measures the failure rates and sends them via telemetry to ground. The exact measurements of failure rates should correspond to those used in the former studies of [2] in order to gain additional insights into the difference to the results from the particle accelerators.

5 Measurements of scalability

One of the key features of *ScOSA* is its scalability. A scalable system is a system which can operate on a large amount of resources as well as on a small amount. In the context of a middleware for spacecraft OBCs, this means that the middleware is capable of operating small-sized spacecraft OBCs, e.g., with only two computing nodes, as well as a system consisting of a large number of interconnected computing nodes.

When introducing more and more computing nodes into a system operated by a middleware, the middleware needs to deal with more participants. This in turn might lead to an increased run-time of the key components of the middleware, as the task mapping or the reconfiguration. In addition to the problem of an increased run-time, the network experiences a higher load caused by the additional participants. This leads to more messages being sent and processed by the recipients, which leads to an increased execution time in the network stack. Those effects on the execution time of the middleware inevitably effect the execution time of the payload applications. As a result, some of those applications may be no longer able to meet their deadlines. This limits the scalability of the system.

5.1 Experiment setup

To obtain and estimate the effects on the system when scaling ScOSA, we conducted experiments with different numbers of nodes virtually on a desktop computer. The scenarios ranged from two nodes up to 20 interconnected nodes. We expected that the scaling affects features of the middleware, as for example the reconfiguration, which in turn then affects the tasks executed by the middleware on the nodes. Therefore, we triggered a reconfiguration by causing a single node-stop failure in each of the scenarios and obtained the time needed for reconfiguring all tasks on the remaining nodes as well as the amount of network traffic caused by the reconfiguration mechanism.

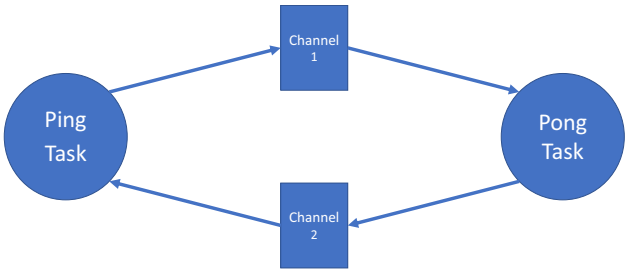

Fig. 6 A self-sustaining task cycle. The experiment application is composed of 20 such task cycles

We also configured the experiments to execute a simple payload application inside the ScOSA middleware. The payload application consists in all experimental runs of 20 independent pairs of tasks (one called "Ping" and one called "Pong"). Those two tasks build, together with two channels, a self-sustaining trigger cycles (see Fig. 6). To heavily load the network, it has been required for each application in an experimental run to assign the tasks of ping-pong pairs to different nodes such that the work load is balanced (see Fig. 7).

To compute the network traffic, the data size of each packet sent by the reconfiguration mechanism was measured. We define the reconfiguration time as the time from the recognition of a node-stop failure by the *coordinator* node until all remaining nodes have applied the new configuration. To mitigate timing errors caused by the operating system of our virtual nodes, we repeated each experiment five times and calculated the arithmetic mean value of those runs. The variances of the different runs were small enough to be neglected. Thus, we only present the mean values here.

5.2 Network traffic

The network traffic of the reconfiguration of the 19 different experiment runs (see Fig. 8) shows a clear linear growth.

This is an expected result as the *coordinator* node informs all the other nodes in the system about the change in the configuration. Subsequently, the nodes inform the *coordinator* node about the applied change. Hence, as the number of nodes increases, the network traffic rises as well.

5.3 Reconfiguration time

Running all the experiments shows a linear but unsteady growth for the obtained reconfiguration time (see Fig. 9), with a three times higher reconfiguration time in the

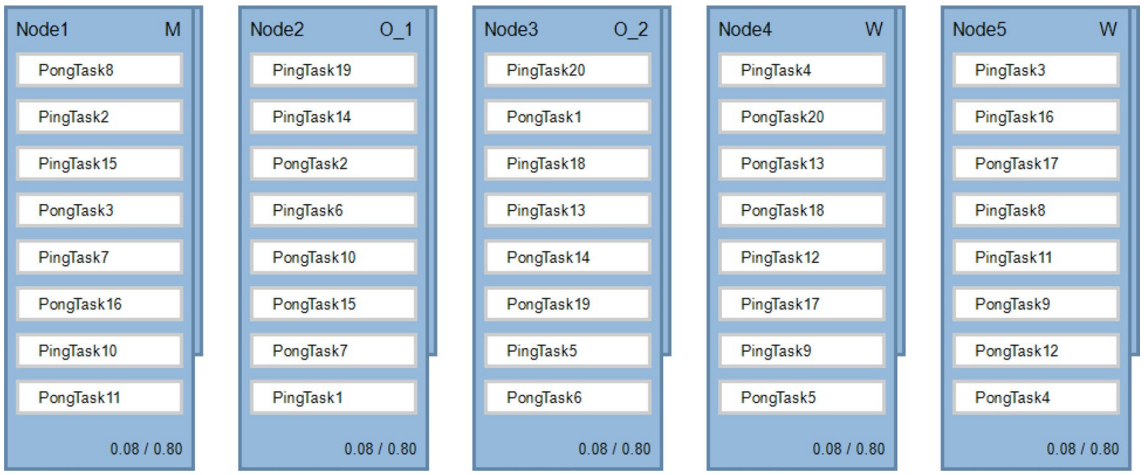

Fig. 7 The tasks are evenly distributed among the nodes of the five node experiment

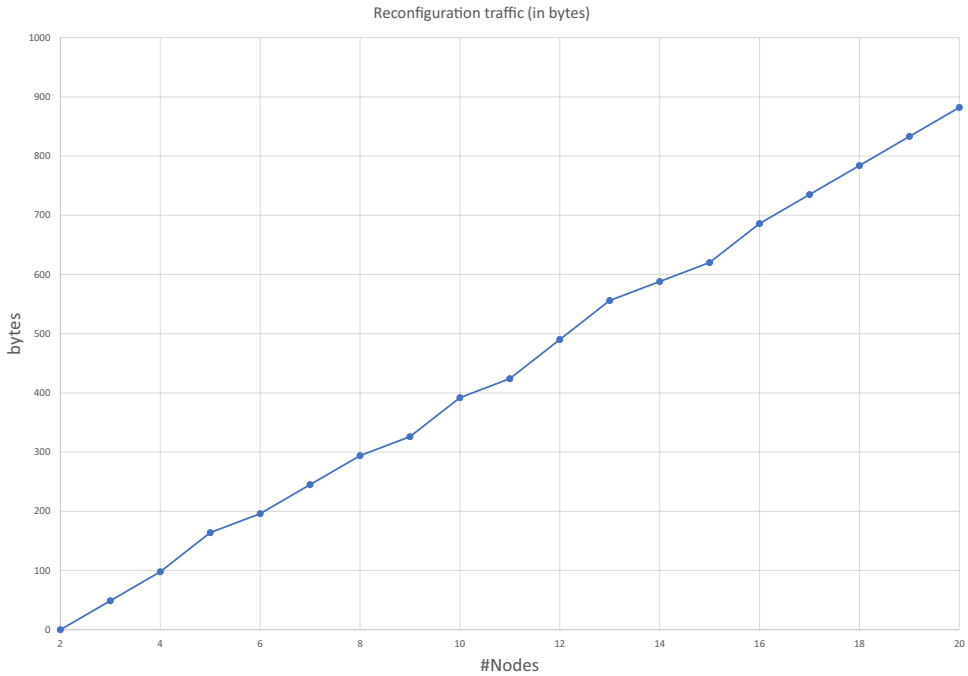

Fig. 8 The network traffic caused by the reconfiguration mechanism of the different experiment scenarios (node count = 2...20)

largest (20 nodes) compared to the smallest (2 nodes) network. The anomalies might be attributable to the scheduling effects of the desktop operating system, on which the virtual nodes were executed. The experiments show that the ScOSA middleware is capable of scaling from 2 to 20 nodes with a linear growth in network traffic as well as reconfiguration time. This shows that on the one hand, the reconfiguration mechanism is generally capable of handling a scaled number of nodes and on the other hand, its overhead scales at a modest, linear rate. The next step is to repeat the experiments on a distributed embedded system.

6 Summary and outlook

Developing a reliable OBC for spacecraft which is able to process complex algorithms fast remains a major challenge. In this paper, we presented the middleware of ScOSA to meet this challenge by abstracting a distributed and heterogeneous architecture consisting of high-performance nodes combined with reliable nodes.

We gave an overview of the middleware and its abilities and explained the different software components comprising the middleware. The middleware originates from two

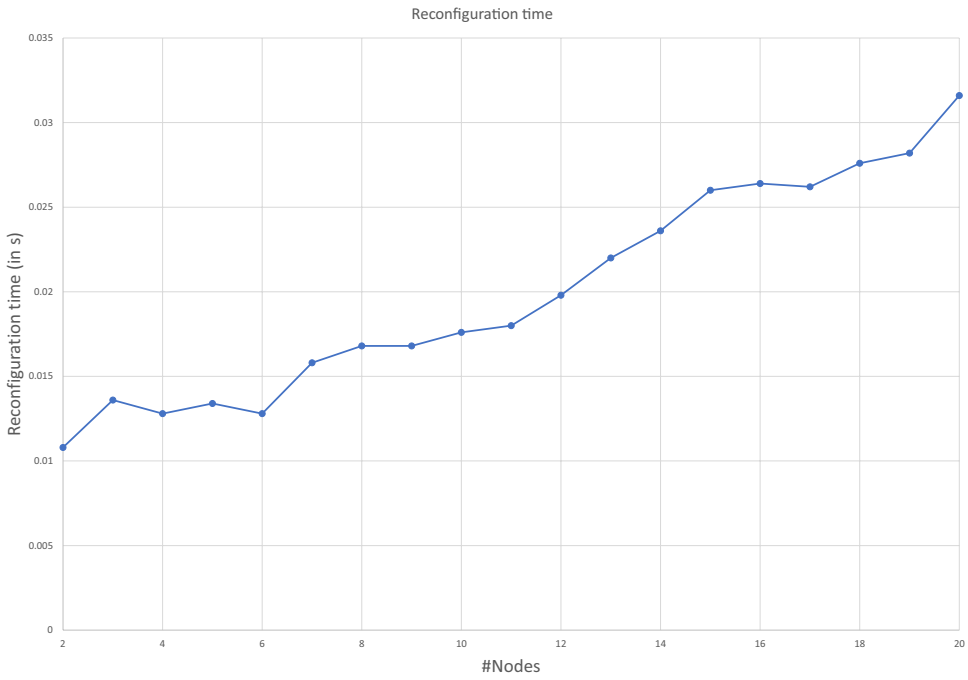

Fig. 9 The reconfiguration time of the 19 different experiments (node amount = 2...20)

projects concerning the issue of high-performance, reliable OBCs, OBC-NG and ScOSA by *DLR*. In the successor project *ScOSA Flight Experiment*, which we briefly presented, the middleware will be further extended and finally demonstrated in-orbit. Before the middleware will become flight-ready, we would like to elaborate some goals in technical and scientific context. We introduced those goals and provided first ideas for the realization of them. Besides that we presented the results of a first virtual scaling experiment, which showed that the overhead of one of the middleware's feature scales with a linear factor. The in-orbit demonstration planned as a secondary payload on the next compact satellite in 2024 will show that future OBCs will consist of a combination of reliable nodes and high-performance nodes to cope with the challenging requirements of future spacecraft missions.

Funding Open Access funding enabled and organized by Projekt DEAL.

References

1. Afonso, F., Silva, C., Tavares, A., Montenegro, S.: Application-level fault tolerance in real-time embedded systems. In: 2008 International Symposium on Industrial Embedded Systems, pp. 126–133 (2008)
2. Amrbar, M., Irom, F., Guertin, S.M., Allen, G.: Heavy ion single event effects measurements of Xilinx Zynq-7000 FPGA. In: 2015 IEEE Radiation Effects Data Workshop (REDW), pp. 1–4. IEEE (2015)
3. Bae Systems PLC: RAD750 radiation-hardened powerpc microprocessor. https://www.baesystems.com/en/download-en/20190103202640/1434555668211.pdf (2008)
4. Balasubramanian, J., Gokhale, A., Schmidt, D.C., Wang, N.: Towards middleware for fault-tolerance in distributed real-time and embedded systems. In: "istributed Applications and Interoperable Systems, pp. 72–85. Springer Berlin Heidelberg, Berlin, Heidelberg (2008)
5. Benninghoff, H., Borchers, K., Börner, A., Dumke, M., Fey, G., Gerndt, A., Höflinger, K., Langwald, J., Lüdtke, D., Maibaum, O., Peng, T., Schwenk, K., Weps, B., Westerdorff, K.: OBC-NG concept and implementation. Tech. rep., Deutsches Zentrum für Luft- und Raumfahrt (DLR) (2016). URL https://elib.dlr.de/102354/
6. Budroweit, J., Mueller, S., Jaksch, M., Alía, R.G., Coronetti, A., Koelpin, A.: In-situ testing of a multi-band software-defined radio platform in a mixed-field irradiation environment. Aerospace **6**(10), 106 (2019)
7. Deutsches Luft- und Raumfahrtzentrum: Die Mission Eu:CROPIS. https://www.dlr.de/content/de/artikel/missionen-projekte/eucropis/mission.html. [Online; retrieved: 2020-11-19]
8. Dorow, K.: Flexible fault tolerance in configurable middleware for embedded systems. In: Proceedings 27th Annual International Computer Software and Applications Conference. COMPAC 2003, pp. 563–569 (2003)
9. Dubey, A., Emfinger, W., Gokhale, A., Karsai, G., Otte, W.R., Parsons, J., Szabó, C., Coglio, A., Smith, E., Bose, P.: A software platform for fractionated spacecraft. In: 2012 IEEE Aerospace Conference, pp. 1–20 (2012)

10. Fayyaz, M., Vladimirova, T., Caujolle, J.: Adaptive middleware design for satellite fault-tolerant distributed computing. In: 2012 NASA/ESA Conference on Adaptive Hardware and Systems (AHS), pp. 23–30 (2012)
11. Hammadeh, Z.A.H., Franz, T., Maibaum, O., Gerndt, A., Lüdtke, D.: Event-driven multithreading execution platform for real-time on-board software systems. In: 15th annual workshop on Operating Systems Platforms for Embedded Real-Time applications, pp. 29–34 (2019). URL https://elib.dlr.de/128249/
12. Hecht, M., Hecht, H., Shokri, E.: Adaptive fault tolerance for spacecraft. In: 2000 IEEE Aerospace Conference. Proceedings (Cat. No.00TH8484), vol. 5, pp. 521–533 (2000)
13. Intel Corporation: Transactional Synchronization with Intel® Core™ 4th Generation Processor. https://software.intel.com/content/www/us/en/develop/blogs/transactional-synchronization-in-haswell.html. [Online; retrieved: 2020-10-10]
14. Katona, Z., Aust, T., Donner, A., Bischl, H., Brandt, H.: Gereleo-modulos-a versatile leo-modem for data relay satellite systems. In: ESA Workshop on Advanced Flexible Telecom Payloads (2016)
15. Keymeulen, D., Shin, S., Riddley, J., Klimesh, M., Kiely, A., Liggett, E., Sullivan, P., Bernas, M., Ghossemi, H., Flesch, G.: High performance space computing with system-on-chip instrument avionics for space-based next generation imaging spectrometers (ngis). In: 2018 NASA/ESA Conference on Adaptive Hardware and Systems (AHS), pp. 33–36. IEEE (2018)
16. Kottmeier, S., Hobbie, C.F., Orlowski-Feldhusen, F., Nohka, F., Delovski, T., Morfill, G., Grillmayer, L., Philpot, C., Müller, H.: The EU:CROPIS Assembly, Integration and Verification Campaigns: Building the first DLR Compact Satellite. In: International Astronautical Congress IAC 2018 (2018). URL https://elib.dlr.de/122106/
17. Lesea, A., Koszek, W., Steiner, G., Swift, G., White, D.: Soft error study of ARM SoC at 28 nanometers. In: Proceeding of the IEEE Workshop on Silicon Errors in Logic-System Effects (2014)
18. Lyons, R.E., Vanderkulk, W.: The use of triple-modular redundancy to improve computer reliability. IBM J. Res. Dev. **6**(2), 200–209 (1962)
19. Lüdtke, D., Westerdorff, K., Stohlmann, K., Börner, A., Maibaum, O., Peng, T., Weps, B., Fey, G., Gerndt, A.: OBC-NG: Towards a reconfigurable on-board computing architecture for spacecraft. In: 2014 IEEE Aerospace Conference, pp. 1–13 (2014)
20. (OMG), O.M.G.: Common object request broker architecture (corba). https://www.omg.org/spec/CORBA/About-CORBA/ (2020). [Online; retrieved: 2020-07-04]
21. Rennels, D.A.: Architectures for fault-tolerant spacecraft computers. Proc. IEEE **66**(10), 1255–1268 (1978)
22. Ricci, M.: Mini-EUSO: a precursor mission to observe Atmosphere and Earth UV emission from the International Space Station. In: ICRC, vol. 301, p. 440 (2017)
23. Schwenk, K., Ulmer, M., Peng, T.: ScOSA: application development for a high-performance space qualified onboard computing platform. In: Huang, B., López, S., Wu, Z. (eds.) High-Performance Computing in Geoscience and Remote Sensing VIII, vol. 10792, pp. 32 – 44. International Society for Optics and Photonics, SPIE (2018). https://doi.org/10.1117/12.2325548
24. SpaceWire Standard: ECSS–Space Engineering." spacewire–links, nodes, routers and networks. Tech. rep., ECSS-E-ST-50-12C (2008)
25. Tambara, L.A., Akhmetov, A., Bobrovsky, D.V., Kastensmidt, F.L.: On the characterization of embedded memories of zynq-7000 all programmable soc under single event upsets induced by heavy ions and protons. In: 2015 15th European Conference on Radiation and Its Effects on Components and Systems (RADECS), pp. 1–4. IEEE (2015)
26. Treudler, C.J., Benninghoff, H., Borchers, K., Brunner, B., Cremer, J., Dumke, M., Gärtner, T., Höflinger, K.J., Lüdtke, D., Peng, T., Risse, E.A., Schwenk, K., Stelzer, M., Ulmer, M., Vellas, S., Westerdorff, K.: ScOSA - scalable on-board computing for space avionics. In: IAC 2018 (2018). URL https://elib.dlr.de/122492/
27. Wilson, C., George, A.: Csp hybrid space computing. J. Aerospace Inform. Syst. **15**(4), 215–227 (2018)
28. Xilinx Inc.: Zynq-7000 SoC Data Sheet: Overview. https://www.xilinx.com/support/documentation/data_sheets/ds190-Zynq-7000-Overview.pdf (2018). [Online; retrieved: 2020-10-10]
29. Yang, W.T., Yin, Q., Li, Y., Guo, G., Li, Y.H., He, C.H., Zhang, Y.W., Zhang, F.Q., Han, J.H.: Single-event effects induced by medium-energy protons in 28 nm system-on-chip. Nucl. Sci. Tech. **30**(10), 151 (2019)
30. Zucherman, A.P., Samson, J.R., Malphrus, B.K.: High Performance Computing Applications in Space with DM Technology. In: 2019 IEEE Aerospace Conference, pp. 1–9 (2019). https://doi.org/10.1109/AERO.2019.8741545